Unified Modeling Language

Peter Fettke
Johannes Gutenberg-University Mainz, Germany

INTRODUCTION

Mature engineering disciplines are generally characterized by accepted methodical standards for describing all relevant artifacts of their subject matter. Such standards not only enable practitioners to collaborate, but they also contribute to the development of the whole discipline. In 1994, Grady Booch, Jim Rumbaugh, and Ivar Jacobson joined together to unify the plethora of existing object-oriented systems engineering approaches at semantic and notation level (Booch, 2002; Fowler, 2004; Rumbaugh, Jacobson, & Booch, 1998). Their effort led to the Unified Modeling Language (UML), a well-known, general-purpose, tool-supported, process-independent, and industry-standardized modeling language for visualizing, describing, specifying, and documenting systems artifacts. *Table 1* depicts the origin and descent of UML.

UML is applicable to software and non-software domains, including software architecture (Medvidovic, Rosenblum, Redmiles, & Robbins, 2002), real-time and embedded systems (Douglass, 1998), business applications (Eriksson & Penker, 2000), manufacturing systems (Bruccoleri, Dieaga, & Perrone, 2003), electronic commerce systems (Saleh, 2002), data warehousing (Dolk, 2000), bioinformatics (Bornberg-Bauer & Paton, 2002) and others. The language uses multiple views to specify system's structure and behavior. The recent version UML 1.5 supports nine different diagram types. *Table 2* and *Figure 1* overview the main concepts of each diagram, a more detailed description is given below. For a full description of all semantics see Fowler (2004), OMG (2003a) and Rumbaugh et al. (1998).

The specification of the UML is publicly available and maintained by the Object Management Group (OMG). OMG's standardization process is formalized and consists of several proposal, revision, and final implementation activities (Kobryn, 1999, p. 31f.). Modeling tools supporting the development of UML diagrams are available from a number of commercial vendors and the open source community (OMG, 2004; Robbins & Redmiles, 2000).

BACKGROUND

There is a great deal of terminological confusion in the modeling literature. A modeling language or grammar provides a set of constructs and rules that specify how to combine the constructs to model a system (Wand & Weber, 2002, p. 364). It can be distinguished between an abstract syntax and a concrete syntax or notation of a language. While the abstract syntax specifies conceptual relationships between the constructs of the language, the concrete notation defines symbols representing the abstract constructs. In contrast, a modeling method provides procedures by which a language can be used. A consistent and suited set of modeling methods is called a methodology. A model is a description of a domain using a particular modeling language.

The UML specification provides an abstract syntax and a concrete notation for all UML diagrams as well as an informal description of the constructs' semantics. The UML's language specification is independent of but strongly related to other OMG standards such as Common Data Warehouse Model, XML Metadata Interchange or Meta Object Facility. A modeling method or a modeling methodology is not defined by the UML standard. Hence, the language is process-neutral and can be used with different software development processes.

Conceptual modeling has a long history. Other modeling approaches that are to a certain degree accepted in practice, for instance the Entity-Relationship Model or flow charts, have a much more limited scope than UML. These approaches address just some aspects of systems' specification, namely data and process view. In contrast, UML supports the specification of static as well as dynamic aspects. Other approaches with a similar scope, for example, Open Modeling Language (Firesmith, Henderson-Sellers, & Graham, 1998), are not widely accepted in practice.

STRUCTURAL DIAGRAMS

Structural or static diagrams describe the objects of a system in terms of classes, attributes, operations, relationships, and interfaces.

(1) *Class diagram*. A class diagram can be viewed as a graph of several elements connected by static relationships. The main element is a class. Classes represent concepts within the system being modeled and are descriptors for a set of objects with similar structure, behavior, and relationships. An

Table 1. History of UML (Fowler, 2004, pp. 151-159; Kobryn, 1999, p. 30)

Version	Year	Comments
0.8	1995	Origin of UML, so-called "Unified Method"
0.9	1996	Refined proposal
1.0	1997	Initial submission to OMG
1.1	1997	Final submission to OMG
1.2	1998	Editorial revision with no significant technical changes
1.3	1999	New use case relationships, revised activity diagram semantics
1.4	2001	Minor revisions, addition of profiles
1.5	2003	Adding action semantics
2.0	2004 (?)	Planned major revision, deep changes to meta-model, new diagram types

Table 2. UML diagram types

Focus	Diagram	Purpose	Main Concepts
Static diagrams	Class	Object structure	Class, features, relationships
	Object	Example configuration of instances	Object, link
Dynamic diagrams	Use case	User interaction with system	Use case, actor
	Sequence	Interaction between objects emphasizing sequences	Interaction, message
	Collaboration	Interaction between objects emphasizing collaborations	Collaboration, interaction, message
	Statechart	Change of events during object's lifetime	State, transition, event, action
	Activity	Procedural and parallel behavior	State, activity, completion, transition, fork, join
Implementation diagrams	Component	Structure and connections of components	Component, interface, dependency
	Deployment	Deployment of components to nodes	Node, component, dependency

object represents a particular instance of a class. Each class has a unique name among other classes within a specific scope (usually a UML package). A class can hold several attributes and operations. Attributes have names and belong to particular types that can be simple data types such as integer, string, and Boolean, as well as complex types (e.g., other classes). Operations are services offered by an instance of the class and may be requested by other objects during run-time. Different relationships between classes can be defined.

Figure 2 depicts a class diagram for banking systems. An account is described by the attributes "number" and "balance." The operations "deposit," "withdrawal," and "freeze" are offered by an account. Each account is kept by a "branch" and is assigned to a "holder." The classes "deposit account" and "current account" reuse the structure and behavior of the class "account" (inheritance relationship). In addition, the specialized account classes define further feature; for example, an object of the class "current account" is described by the property "overdraft facility" and offers an operation calculating the current debit balance.

(2) *Object diagram*. An object diagram is an instance of a class diagram and depicts the state of the system at a point in time (e.g., a particular configuration of several objects). It contains objects including their actual values of attributes and links describing object references.

BEHAVIORAL DIAGRAMS

Behavioral diagrams describe the dynamics between objects of a system in terms of interactions, collaborations, and state histories.